Technology that talks business

Personalization at scale

Nick Hills // Sitecore MVP // nick.hills@trueclarity.co.uk
- 25 April 2016

Contents

Keeping things simple

Before trying to work out your dynamic content strategy it's worth thinking about a slightly simpler model – non-personalized static content.

Consider content stored within Sitecore; it's not going to be static. Editors will be making changes and depending on your publishing strategy will be pushing this content periodically to the customer facing site. This introduces an interesting challenge – how do you ripple out these changes to all consumers? The more copies of the data that are stored the more places that need updating when publishes complete.

The table below shows a variety of approaches & options for scaling up how you choose to distribute content. *Note, the pros and cons don't consider things like resiliency, DR or failover.*

Technology / approach	- Pros	- Cons
CDN's Selected content is proxied and edge cached by a CDN provider.	- Globally distributed content: improves latency and performance for customers - Device optimized content - Reduces load onto your origin	- Decaching post publish requires feedback loop with CDN - Your CDN provider imposes the criteria and options for cache keys - Purging item by item from the cache requires additional logic
Interim content repository An interim repository is used to cache and manage the state of content. Edge applications consume content from this repository.	- Content can be cached in stores specifically scaled and designed for purpose - These stores can be globally distributed	- Requires custom development to both publishing, consumption and the repository itself - Decaching post publish requires feedback loop - Additional kit required and comes at a price - Sitecore is a content repository, this duplicates its function
Custom caches Content is served from custom caches. Publishes explicitly clear the caches.	- You have full control over what is being cached and for how long - Decaching post publish can be built in	- Requires custom development - Needs server resources to run & store the cached data - Purging specific items requires additional logic
Sitecore caches Out the box Sitecore caches. These store things like data, items and html	- Out the box offering within Sitecore - Automatically updated post-publish	- Capacity limited by the size of the boxes - Needs server resources to store the cached data

	- Partial decaching out the box	- Vulnerable to application restarts
No caches	- A brave move – there are very few advantages of this approach	- As load increases you have no guard against your resources e.g. databases will come under more and more load
Render content to disk At the point content is published you generate a static version which is then served to the user	- Content can be served quickly off disk - Databases and other content aggregation resources will subsequently receive a lot less load - Tolerant to application restarts	- Custom functionality required during publishing to generate the files - Disk space required to store all the content - Extra load on the system that generates the files - Purging strategy required for old entries - Decaching (after publishes) requires purging strategy
Vertical scaling Add more grunt – bigger boxes, more CPUs, more RAM, faster and more disks	- Larger boxes can handle more requests - No additional server licenses required	- Additional kit comes at a price - Bigger webservers also need bigger databases. There will eventually be a limit that a single database server can handle.
Horizontal scaling Add more boxes, or clusters of boxes – this would require boxes for the application and databases.	- More boxes can handle more requests - Easy to pull clusters in and out of the customer facing infrastructure for things like deployments & upgrades	- Publishing content to more clusters requires either more publishing targets or alternative content distribution channels - Additional kit comes at a price - More server licenses required

TABLE 1 – PROS AND CONS OF DIFFERENT SCALING OPTIONS

Let’s get personal

What is a personalized page? Chances are this will vary client by client, project by project and even page by page.

During a request we know something about the user. This could be as detailed as every single page view and order summary through to simply the presence of a cookie or querystring parameter. This information is then used to influence the composition of the page.

What do you need to personalize a page?

Before deciding where and how to compose the pages its worth considering the make-up of a personalized page.

The content

Sitecore’s whole foundation is built around being a Content Management System so this should be pretty simple. This could be e.g. html, images, scripts, styles and more.

Information about the user

During the scope of a request we’ve identified the current user. What else do we know about them? *They phoned the help centre to ask about a new zoom lenses. They requested a brochure on high-end cameras. They browsed the high-end lens section of the site.* xDB contains, or at least can contain all this information.

The rules

Why should we show high-end lens adverts to one user and compact camera adverts to the next? We have information about the user and therefore need to run some rules against this data. In its simplest form this could be something like:

If (user.HighEndCameras == true) then content = ‘Have you seen our latest high end lens. Click here for more’.

Note, this is an over simplification however highlights the fundamental concept of the rules.

Where to show the content?

In the lifecycle and construction of a Sitecore page, components are configured and positioned via **placeholders**, **datasources** and **rendering parameters**. This is why you might see adverts to the right of the content. The content is in one placeholder, the adverts another.

When to show the content?

We have the content, we have the knowledge of the user and we have the rules. Therefore if we evaluate the three together we know when to show our user their high-end lens advert.

There is one scenario: what if none of the rules evaluate? You could either make no changes to the page or fallback to default content.

How to show the content?

The content is no different to any other content on the site. It will contain text, links, images etc. so can be rendered in the same manner as the rest of the page.

How to enrich the content?

This really depends on your approach and the type of data you need to display. An example would be: *Sitecore is the master repository for advert content. All the equipment prices come from another external system. When do we aggregate the content and prices so that the advert shows: '**Check out our latest high-end lens. <u>Now only £941</u>**'.*

This kind of customization adds yet another parameter to the variability of the data. Not only is the content user specific but also varies based on another external system. This could have its own rate of variance e.g. prices are refreshed and re-imported every N minutes.

Where should page composition occur?

Typically content will flow from the server – in our case served by an asp.net Sitecore application. If your application is configured to use them, through a set of proxies & CDN's and finally ends up at the client's browser.

In Sitecore the default behavior for composing personalized content is at the server. Content is aggregated, rendered and finally sent to the client as a full page.

Where to run the rules?

In order to evaluate that our user sees adverts tailored to their needs we need to run some rules. The default behavior within Sitecore is that these are evaluated at the server during the page composition and rendering phase of a request.

Let’s consider the options:

At the server per request

Each page load you gather information about the current request and current user. Based on the request the page layout and component configurations are retrieved. If any component requires personalization its rules are evaluated. If the rules require user information this is retrieved from xDB & your xDB session provider.

Pros:

- This is out the box Sitecore behavior.

Cons:

- Functionality is coupled to the performance of an ever growing xDB mongo repository.
- It’s tricky to slice the application in order to layer caching. C*aching certain layers of the application offers the benefit that performance isn’t compromised under high load.*
- Turning off personalization under load is not trivial.
- Edge-cachability of the page becomes tricky. *Note, this depends on the number of permutations of the page that the rules evaluate to. If this is a small subset then edge-caching per permutation is feasible. However if this varies for every user then edge-caching isn’t recommended.*

Pre-generate the permutations

Sitecore offers several options for detecting when content has been saved or published. An example would be the **item:saved** event. When content is saved or published you generate the permutations of the widgets, or even better pages upfront. When users then request these pages they have already been personalized.

Pros:

- You don’t have to evaluate the personalization rules every request. However you still need to select the correct outcome for the current user.

Cons:

- You need to store all the permutations of the outcomes – be it components or pages.
- You need to generate all the permutations for all combinations of users. The more parameters - the more permutations.
- As the content gets more personal e.g. data specific to just one user, the number of permutations rockets.
- The more permutations stored the more entries that need invalidating or updating post publish.
- Something needs to generate all these component snippets or pages. Post re-publish this set could be huge and thus would put high strain on the application responsible for generation.

Distributed personalization

The responsibility for personalizing content is removed from the core content application. Examples could be: custom external services, custom proxies, external providers or even CDN's.

Pros:

- Can be tailored for specific needs in terms of capacity and performance.
- Easier to turn off as less baked into the core application.
- If a CDN is used, global distribution becomes easier.

Cons:

- The more duplicates of the data the harder de-caching becomes post-publish.
- You are potentially holding configuration data in more than just the CMS. Admin users would need to know where to edit content vs rules vs presentation logic
- Requires a large amount of custom development.

At the client

Rather than composing the page and associated components at the server you move as much as possible to the client. Each aspect of the personalization is dissected and exposed by the server. This is then consumed and composed at the client.

Pros:

- Serving the initial base page can reap all the rewards that CDN's or custom edge caching can offer.
- Not only can the base page be edge cached, the same applies to the feeds that serve the content and the rules.

- Each application component (base html, rule & content feed, contact feed) can be scaled and tuned independently.
- The client is doing the work and evaluating the rules. The origin has even less to do.
- We can turn off the personalization when required.
- If issues arise we can guard against page composition errors via things like circuit breakers over the feeds.

Cons:

- By moving the rules to the client you are putting more work in the hands of the device. Older less powerful mobile devices will take longer to evaluate the rules than a high-spec desktop machine.
- Updates become trickier as you need to keep the client store that holds the Contact information in sync with xDB.
- Depending on the approach taken the number of ajax requests can grow as you personalize more of the page. *However, this could be designed so that things are batched into groups of content & rule requests.*
- You need to build this approach on top of Sitecore and requires custom coding.
- Certain information isn't accessible from javascript e.g. request headers. You get a rich set of parameters available in javascript: cookies, url and querystring, local storage etc. but certain parameters aren't accessible. Most of these could be worked around by moving the scope of where the parameters and their associated content exists. *Note, to ensure the rules are javascript compliant you can create custom rule groups – this ensures that only selected rules are available to the editors.*
- The default behavior for logging visits and interactions will be wrong as all the requesting urls will be the feeds and not the source pages.
- Triggering page events and goals will require custom implementations to handle the flow of data into xDB.
- Your client implementation will need to handle the behavior of what to do if no rules match, the simplest being no change is made to the base html.

What to store in each scenario?

There are several options for where to store things, in each scenario what do we want to store?

The following criteria are worth considering:

- The number of permutations of the output & storage requirements.
- The complexity of the option.
- The overhead required to generate permutations.
- The work still required to select and then render the output based on the request and knowledge of the user.
- The ease of updates post-publish and during user interaction.
- The range of parameters available for personalization.

Location	**- Pros**	**- Cons**
Full page A full version of every page and its permutations is composed & stored	- Can be edge-cached as long as cache provider can determine which variant to serve to each user	- Has the highest number of output permutations - Server side page construction required
Full page and then by widget A full version of the underlying page is composed with cached versions of each personalized widget	- Out the box Sitecore behavior	- Has a high number of output permutations - Server side page construction required
Segmented view of the data A custom cache stores a subset of permutations of every personalized page. This subset is based on high-level segments of the data	- Edge caches can globally distribute the page variants	- Limited scope for personalization
Feeds and cookies Base html is served to the client. Cookie information then provides keys to access feeds of cookie specific rules and content.	- No base page blocking - External factors can influence personalization e.g. cookies updated elsewhere dictate behavior - Feeds can be cached	- Limited scope for personalization - Custom code required - Restriction on scope of rules e.g. access to headers
Feeds and client composition Base html is served to the client. Cached xDB information then provides keys to access feeds of specific rules and content.	- No base page blocking - Full xDB contact knowledge can be used - Each feed can be scaled and in the most-part cached - External factors can influence personalization e.g. cookies updated elsewhere dictate behavior	- Custom code required - Restriction on scope of rules e.g. access to headers - Updates require sync'ing to both the client and server

TABLE 2 – WHAT TO STORE IN EACH SCENARIO

How to pick what's best for you?

Before we answer that it's worth considering the key factor that influences a lot of the different options above: **parameters**. Now that might sound trivial however plays a big part in deciding which option, or set of options is best for you. It's also important to understand that not one size fits all – different areas of your site may well benefit from a mixture of the approaches we've discussed.

How can parameters be so important?

I'd encourage you to think of the whole system; what goes in and what comes out.

- How do things vary for one user compared to another?
- How granular are the rules that run – does every user see a different result?
- How do other factors influence the system e.g. prices changes invalidating caches; publishes leading to content de-caching?
- What impact does language have on the setup and the number of permutations?
- Which aspects of the system understand the difference between each user?
- If relevant, where should currency be used and how does that vary in relation to language?
- How often does the data about each user change and what can update it?
- How many users will the system contain?
- How many permutations of each widget will there be?
- Does personalization just affect things like adverts or can it influence a user's journey?

The list goes on! Undoubtedly it will vary site by site, even page by page and so is important to consider.

tl;dr

There are too many options, where do I start? Initially keep it simple and be realistic.

If you are anticipating every user receives a fully customized page by page experience you need to expect more overhead generating all the permutations - chances are this will also reduce the pages edge-cachability. However if you keep things simpler and play to your strengths the out the box Sitecore functionality will go a long way.

As your traffic increases ensure you provide the infrastructure to match and don’t be scared to shuffle things around. Remember, the more you can cache the better. A solution for personalizing adverts in a carousel doesn’t necessarily help tailor a user’s journey through a check-out process.

Remember: Consider the parameters!

About

Nick’s worked at True Clarity for over 10 years and been a Sitecore MVP since 2012. During that time he’s worked on a wide range of Sitecore and non-Sitecore clients e.g. Asos, easyJet, Dyson, Sophos and many more.

You can find out more online at: http://blog.boro2g.co.uk and http://www.trueclarity.co.uk